Optimizing Synchronous Systems for Multi-Dimensional Applications*

Nelson L. Passos and Edwin H.-M. Sha

Dept. of Computer Science & Eng.
University of Notre Dame
Notre Dame, IN 46556

Liang-Fang Chao

Dept. of Electrical & Computer Eng.
Iowa State University
Ames, Iowa 50011

Abstract

Time-critical sections of multi-dimensional problems, such as image processing applications, are in general iterative or recursive. In this paper these sections are modeled as cyclic multi-dimensional data flow graphs (MDFGs), which are also used to represent the digital circuit designed to compute such problems. Each node in the MDFG is associated with a set of functional elements in the circuit. Memory elements and circuit paths are associated with graph edges representing data dependencies. This new optimization technique consists of a multi-dimensional retiming being applied to the MDFG to reduce its cycle time while considering memory requirements. This technique guarantees that all functional elements of a circuitry, designed to be applied to problems involving more than one dimension, can be executed simultaneously. The algorithm runs in $O(|E||V|)$ time, where V is the set of nodes and E is the set of edges of the MDFG representing the circuit.

1 Introduction

Computation-intensive applications usually depend on time critical sections, consisting of loops of instructions also called iterations. The design of application-specific solutions for such sections improves the computing performance. A multi-dimensional (MD) retiming transformation is presented in this study to achieve an additional improvement, while considering the consequences of such transformations on the memory requirements.

Retiming was initially proposed by Leiserson-Saxe [4] focusing on 1-D problems. Such a technique presents a lower bound in the achievable cycle time due to the the number of delays existing in a cycle. Most of the research in this area has followed this approach and consequently subject to the same constraint [2]. Multi-dimensional systems have been covered in studies using linear programming methods [5, 7]. However, the parallelization of operations that require data produced on the same iteration depends on the use of multiple processors. In this paper, we utilize the concept of MD retiming presented in [1, 6] to model the placement of registers along the circuit data paths, while considering the consequences in the memory structure. This new method is applicable to any problem involving more than one dimension.

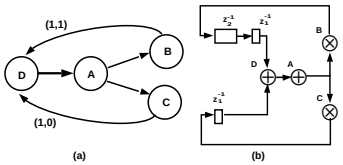


Figure 1: (a) special case of MDFG where row-wise computation is an optimization problem (b) circuit design, notice that z_1^{-1} is equivalent to one register

For simplicity, we use 2-D problems as examples. The two dimensions are generically referred as rows and columns. Let's examine a simple example shown in figure 1, where nodes D, A are adders and B, C are multipliers. Figure 1(b) shows the equivalent digital circuit. We used the notation z_i^{-1} to indicate a delay element in the i direction. If assumed that the computation follows a row-wise sequence and that the total number of points in the row-direction is M, then the 2-D delay $(1, 1)$ can be represented by a FIFO structure of size $M + 1$, i.e., a serial implementation of z_1^{-1} and z_2^{-1} elements. The z_1^{-1} element representing the 2-D delay $(1, 0)$ is equivalent to only one delay. The current cycle time for this design is equivalent to the sequential execution of two additions and one multiplication. In this case, the initial assumption on the computation sequence strongly affects how much the design can be optimized, since the 1-D retiming can not reduce the initial cycle time. By using our algorithm, we apply a 2-D retiming $(-2, 4)$ to node D and $(-1, 2)$ to node A, resulting in the fully parallel solution shown in figure 2 with a cycle time equivalent to one multiplication.

*This work was supported in part by ORAU Faculty Enhancement Award under Grant No. 42265, by the NSF Research Initiation Award MIP-9410080, and by the William D. Mensch, Jr. Fellowship.

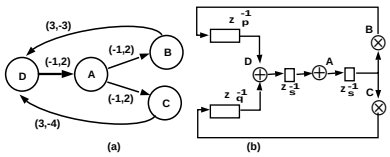


Figure 2: (a) retimed MDFG, ignoring the original computation sequence (b) optimized circuit design

2 Basic Principles

A circuit design is modeled as a *multi-dimensional data flow graph (MDFG)* $G = (V, E, d, t)$, where V is the set of computation nodes, i.e., the functional elements in the circuit design, E is the set of dependence edges, equivalent to the circuit data paths, d is a function representing the multi-dimensional delay, also known as dependence vectors, between two nodes, implicitly indicating the storage elements required in the circuit design, and t is a function representing the computation time of each node. We use $d(e) = (d_x, d_y)$ as a general formulation of any delay shown in a two-dimensional data flow graph (2DFG). An *iteration* is the execution of each node in V exactly once. We say that a design is *fully parallel* if all nodes in one iteration can be executed simultaneously.

Vectors are used to indicate the sequence of computation. A *schedule vector* s is the normal vector for a set of parallel hyperplanes that define such a sequence. Nodes in the same hyperplane will be executed sequentially, according to a second level of schedule associated with the hyperplane. We say that an MDFG $G = (V, E, d, t)$ is *realizable* if there exists a schedule vector s for the iteration space with respect to G, i.e., $s \cdot d \geq 0$ [3] and it has no zero-delay cycle. An *orthogonal schedule vector* is a schedule vector parallel to one of the axis representing the dimension directions in the problem. A *non-orthogonal schedule vector*, requires a wavefront execution sequence.

3 Multi-Dimensional Retiming

An *MD retiming* r redistributes the nodes of an MDFG in the iteration space, such that each iteration still has one execution of each node in G. This transformation is equivalent to a redistribution of the delay elements in a circuit. A retiming vector $r(u)$ represents delay components pushed into the graph edges $u \rightarrow v$, and subtracted from the edges $w \rightarrow u$. The choice of the correct retiming function is important to guarantee the realizability of the circuit. A *legal MD retiming* for an MDFG G transforms G in $G_r = (V, E, d_r, t)$ such that G_r is still realizable. We use a general set of constraints for a legal MD retiming that produces full parallelism among the nodes of an MDFG:
(1) the iteration space of the retimed MDFG G_r does not contain any cycle.
(2) $d_r(e) \neq (0, 0, \ldots, 0)$ for any edge $e \in E$.

In order to enforce these constraints, initially, we propose the utilization of a schedule vector s such that $d(e) \cdot s > 0$ for every $d(e) \neq (0, 0, \ldots, 0)$. It is obvious that such schedule vector exists, whenever the MDFG is realizable. The following theorem introduces the method of computing a legal MD retiming.

Theorem 3.1 *Given an MDFG $G = (V, E, d, t)$, s a schedule vector for G, and $u \in V$ a node with all the incoming edges having non-zero delays. A legal MD retiming $r(u)$ is any vector orthogonal to s.*

Proof: For some node u with incoming edges e_1 and outgoing edges e_2, s.t., $d(e_1) \neq (0, 0, \ldots 0)$,in order to verify if the resulting MDFG is realizable, we compute the inner product of s and each of the retimed dependence vectors. Thus, $d_r(e_1) \cdot s = d(e_1) \cdot s - r(u) \cdot s = d(e_1) \cdot s > 0$. For e_2, $d_r(e_2) \cdot s = d(e_2) \cdot s + r(u) \cdot s = d(e_2) \cdot s$. We have now two cases:
Case 1: if $d(e_2) \neq (0, 0, \ldots 0)$ then $d(e_2) \cdot s > 0$, and the resulting graph is realizable.
Case 2: if $d(e_2) = (0, 0, \ldots 0)$ then $d_r(e_2) = r(u)$. Since $d_r(e_1) \cdot s > 0$ and for each edge e, $d(e) \cdot s > 0$, it is impossible to have a linear combination of these delay vectors orthogonal to s; i.e., parallel to $d_r(e_2) = r(u)$. Therefore, the resulting graph is realizable. □

Here we introduce an important corollary from theorem 3.1. Corollary 3.2 explores the capability of using multiple values of the retiming function.

Corollary 3.2 *Given an MDFG $G = (V, E, d, t)$, s a schedule vector for G, a vector r orthogonal to s, a node $u \in V$ with all incoming edges non-zero, and $k \geq 1$, then $(k \times r)(u)$ is a legal MD retiming.*

4 Memory Size Considerations

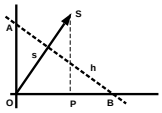
The final schedule vector plays an important role in the MD retiming process. In this section, we discuss the relationship between the schedule vector and the storage elements in the circuit design.

In a two-dimensional (2-D) problem, a row-wise execution is equivalent to a schedule vector $(0, 1)$ and it implies that delays $(1, 0)$ must be translated into single register delay elements. Delays of the form $(d_x, 0)$ produce queues of size d_x, which is independent of the problem size. If M is the number of points in the row-direction, the delays of the form $(0, d_y)$ are equivalent to queues of size $d_y \times M$. Figure 3(a) shows a sequence of execution imposed by a row-wise computation in a two-dimensional problem. We notice the progress in the y-direction as indicated by the schedule vector $(0, 1)$, and a faster recursion in the x-direction. Non-orthogonal schedule vectors define execution hyperplanes that require a more complex formulation of the queue sizes. Because these hyperplanes are not parallel to the orthogonal axis x or y, the number of points vary according to the boundaries of the iteration space and the slope of the hyperplane. The following lemma shows how to compute the

maximum number of integral points in an execution hyperplane for a two-dimensional iteration space with equal number of points in both directions:

Lemma 4.1 *Given a 2DFG G with an iteration space of dimensions $M \times M$, an execution hyperplane h defined by a schedule vector $s = (s_x, s_y)$, the maximum number of integral points $P(h)$ for h is:* $P(h) = \left\lfloor \frac{M}{max(|s_x|,|s_y|)} \right\rfloor$.

Proof: From the geometric construction below, where h is a hyperplane defined by s, the triangles OSP and OAB

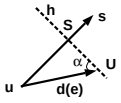


are proportional. Since OS is the distance between two integral points, we may assume OS = 1, then $AB = \frac{OA}{OP} = \frac{OB}{SP} = \frac{OA}{|s_x|} = \frac{OB}{|s_y|}$. But $OA \leq M$ and $OB \leq M$ then $AB \leq \frac{M}{|s_x|}$ and $AB \leq \frac{M}{|s_y|}$. Therefore $AB \leq \frac{M}{max(|s_x|,|s_y|)}$. Using integral points, $AB \leq \left\lfloor \frac{M}{max(|s_x|,|s_y|)} \right\rfloor = P(h)$. □

The distance between hyperplanes, computed by $s \cdot d$ for some dependence d [3], and the maximum number of points in a hyperplane give us a good approximation for the upper bound of the queue size required between the production and consumption of some data. However, dependencies may have a displacement component in the hyperplane direction whenever the dependence vector is not parallel to the schedule vector. This displacement can add or subtract points to that upper bound, depending on the vector direction. To compute the number of points affected by this displacement we use the following lemma:

Lemma 4.2 *Given a 2DFG $G = (V, E, d, t)$ with an iteration space of dimensions $M \times M$, a schedule vector $s = (s_x, s_y)$ for G, $u \in V$, a sequence of execution $s^{\perp} = (-s_y, s_x)$ for each hyperplane h defined by s, and $u + d(e) \in h$, the number of points $P_h(d)$ between $u + d(e)$ and the projection of node u in h is given by:* $P_h(d) = \left\lfloor \frac{s^{\perp} \cdot d(e)}{|s^{\perp}|^2} \right\rfloor$

Proof: From the geometric construction below, where h is a hyperplane for s, the distance SU in integral points



can be computed as $\frac{|d(e)|cos\alpha}{|s^{\perp}|}$. However, we know that $|d(e)|cos\alpha = \frac{s^{\perp} \cdot d(e)}{|s^{\perp}|}$ therefore, $P_h(d) = \left\lfloor \frac{s^{\perp} \cdot d(e)}{|s^{\perp}|^2} \right\rfloor$ where the floor function is used to guarantee the result as an integer. □

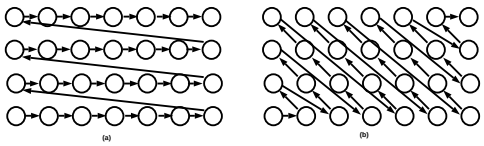


Figure 3: (a) Iteration space and a row-wise execution s=(0,1) (b) Iteration space and a non-orthogonal execution sequence with a schedule vector (1,1)

Following this reasoning, the next theorem shows a general formulation to compute the maximum queue size required by a specific delay vector for 2-D problems:

Theorem 4.3 *Given a realizable 2DFG $G = (V, E, d, t)$, a schedule vector $s = (s_x, s_y)$ for G, and a set of hyperplanes h defined by s, the maximum queue size for some dependence $d(e)$ is given by:* $Q_{d(e)} = (s \cdot d) \times P(h) + P_h(d)$.

Proof: Immediate from lemmas 4.1 and 4.2. □

If $s \cdot d = 0$, contradicting the definition of schedule vectors presented earlier, the size of the queue for those delay vectors would be reduced to the term $P_h(d)$, which represents a fixed size queue not dependent on the problem dimensions. Figure 3(b) shows an iteration space and the execution sequence for a schedule vector $(1, 1)$. Delay vectors with value $(-1, 1)$ represent a dependence within a hyperplane, and are translated into one register at the circuit level. In order to optimize the memory design, we try to select a schedule vector s such that the term $s \cdot d$ is small or eventually zero. Therefore, we redefine the constraints in the selection of the schedule vector by allowing $s \cdot d = 0$. Intuitively, we know that the non-zero delay vectors whose products with s could result zero must be the outermost dependencies in the span of non-zero delay vectors. However, we also know that a first translation of dependencies in registers can introduce the constraints found in the 1-D retiming. Therefore, to avoid such constraints we introduce a new selection criteria for s. We assume, without loss of generality that all nodes represent unit time operations and define such criteria in the following theorem.

Theorem 4.4 *Given a realizable 2DFG $G = (V, E, d, t)$ with $t(v) = 1$ for any $v \in V$, if d_1, d_2 are the outermost delay vectors of G, a suitable schedule vector s for obtaining full parallelism of G through MD retiming is one of the following:*
(1) $s \perp d(e)$, for $d(e) \in \{d_1, d_2\}$ and for any cycle l, if $e \in l$ then $d(l) \neq k \times d(e)$ for any integer k.
(2) $s \perp d(e)$, for $d(e) \in \{d_1, d_2\}$ and for any cycle l, if $e \in l$ then $d(l) = k \times d(e)$ for any integer $k \geq t(l)$.
(3) s such that $d(e) \cdot s > 0$ for any non-zero delay edge $e \in E$.

Proof: To obtain full parallelism implies to distribute the MD delays in such a way that no zero-delay edges are left after retiming. For case (1), we have $\sum_0^n d(e_i) = d(l)$ for

$l = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \ldots \xrightarrow{e_n} v_0$. Without loss of generality, assume the first j edges have zero delay. After retiming, $\sum_0^{j-1} d_r(e_i) = j \times r$ but $\sum_j^n d(e_i) = d(l) \neq k \times r \ \forall k$, then $d(e_i) \neq k \times r \ \forall j \leq i \leq n$, so $d_r(e_i) = d(e_i) - m \times r \neq (0,0)$ for some $0 \leq m \leq j$. Therefore, all edges have non-zero delays. In the second case, $d(l) = k \times d(e)$ and k is greater than the number of edges in the cycle, then there is at least one distribution where $t(l) - 1$ edges have $d_r(e) = r$ and one edge has $d_r(e) = (k - t(l) + 1) \times r$. Case 3 is similar to case 1 since r is not parallel to any d. □

According to the theorem above, the schedule vector is initially selected to be orthogonal to one of the outermost dependence vectors. If the constraining conditions are not satisfied, the schedule vector is selected based on the original definition under the constraint $d(e) \cdot s > 0$. For this last case, we decided to work with a schedule vector $s = (d_1 + \gamma \times d_2)$, where d_1, d_2 are the outermost delay vectors and $\gamma \geq 1$, s.t., s is not orthogonal to d_1 or d_2, since this is a guaranteed choice that results $d(e) \cdot s > 0$. After choosing the schedule vector, we need to keep the retimed vectors equivalent to minimum size queues. We choose a retiming function in such a way to produce minimum delays, as defined below:

Definition 4.1 A *minimum MD retiming function* r is a legal MD retiming such that there is no integral vector f and integer constant g such that $r = g \times f$.

5 The Algorithm

In a first approach for transforming a 2DFG and, consequently, its equivalent circuit into a fully parallel MDFG, our algorithm verifies if there is a negative cycle in the iteration space whenever the retiming vector used is parallel to any of the outermost delay vectors. We call this procedure, shown below, *ParallelRetim*. It is the first section of the main algorithm described later.

ParallelRetim(G)
/* compute the outermost dependence vectors */
$\{d_1, d_2\} \leftarrow$ OUTERMOST (G); $found \leftarrow FALSE$
for each $d \in \{d_1, d_2\}$ and $found = FALSE$ do {
$r \leftarrow \frac{d}{gcd(d_x, d_y)}$ /* find the minimum retiming function */
$\forall e \in E$, if $\left(\frac{d(e)_x}{r_x} = \frac{d(e)_y}{r_y}\right)$ then WEIGHT(e)=$\frac{d(e)_x}{r_x}$
else $E \leftarrow E - \{e\}$
/* modified single-source shortest path follows */
$count \leftarrow 0$
$\forall u \in V$ LENGTH(u) $\leftarrow 0$; PUT($u, Queue$)
$tail \leftarrow$LAST($Queue$)
while $Queue \neq \phi$ and $count < |V|$ {
GET($u, Queue$)
$\forall v$, such that $u \xrightarrow{e} v$
if LENGTH(u) + (WEIGHT(e)) $-$ 1 <LENGTH(v)
LENGTH(v) $\leftarrow$LENGTH(u) + WEIGHT(e) -1
if $v \notin Queue$ then PUT(v,$Queue$)
if $u = tail$ then $count++$, $tail \leftarrow$LAST($Queue$) }
$found \leftarrow (Queue = \phi)$ }
return ($found$)

If the chosen retiming is not valid, a new retiming vector, not parallel to any delay vector, is chosen. We call the main algorithm *Comdr* for *Circuit Optimization via Multi-Dimensional Retiming*. The mathematical description of this algorithm is presented below:

Algorithm Comdr(G)
if (ParallelRetim(G) = FALSE)
/* use vector non parallel to the delay vectors*/
for ($\gamma = 0$; $d \leftarrow (d_1 + \gamma d_2)^{\perp}$ non-parallel to d_1 or d_2; $\gamma++$);
$r \leftarrow \frac{d}{gcd(d_x, d_y)}$ /* find the retiming function */
/* begin topological sort */
$\forall e \in E$, if $d(e) \neq (0,0)$ then $E \leftarrow E - \{e\}$
$count \leftarrow 0$
while $V \neq \phi$ {
$\forall u \in V$ if OUTDEGREE(u) = 0
PUT($u, Queue$); $V \leftarrow V - \{u\}$
while $Queue \neq \phi$ {
GET($u, Queue$); LENGTH(u) $\leftarrow count$
$\forall\ w$, s. t. $w \rightarrow u$
OUTDEGREE(w) $\leftarrow$ OUTDEGREE(w) -1 }
$count++$ }
/* compute the retiming function for each node*/
$\forall v \in V$, compute $r(v)$ =LENGTH(v)$\times r$
end

In the first step, the outermost dependence vectors for a 2DFG are computed in $O(|E|)$ time through the usage of the function OUTERMOST. The first outermost delay vector is considered to be parallel to the retiming vector. Dependence vectors non-parallel to the retiming vector are considered infinite source of delays and are removed from the graph. On the other side, dependence vectors parallel to the retiming function are transformed into 1-D values representing how many times we can retime that edge. Therefore, a typical 1-D retiming solution is used to verify that the retiming is legal. A modified single-source shortest path algorithm computes in $O(|V||E|)$ time the shortest path lengths measured from a single-source node, providing us the coefficients for the final MD retiming function. If the algorithm fails for both outermost edges, the greedy solution is adopted, i.e., $s = (d_1 + \gamma \times d_2)$ which guarantees that a solution is found. A modified topological sort is used to order the nodes in levels, stored in the vector LENGTH, in $O(|E|)$ time. Every node is assigned to a unique level number, producing the coefficients for the MD retiming function. This algorithm guarantees that a fully parallel solution is achievable, and only single registers, or fixed number of registers are inserted into the direct circuit paths. These properties are shown in the theorem below:

Theorem 5.1 *Given an MDFG $G = (V, E, d, t)$, the optimization algorithm Comdr transforms G to G_r, such that;*
(1) G_r is realizable
(2) G_r is fully parallel
(3) all zero delay edges $u \rightarrow v$ of G, after retiming, will have fixed-size queues of size k, $k \geq 1$, when $(k \times r)(u)$ was the retiming function applied to u.

Proof: (1) By using the algorithm Comdr, the single-source shortest path algorithm will detect a cycle if one

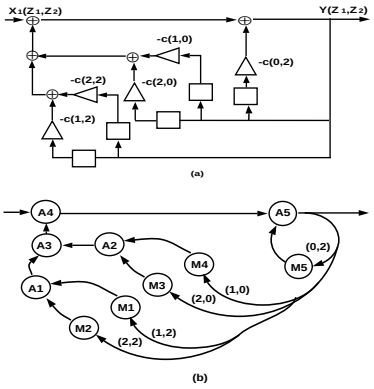


Figure 4: (a) circuit design for an IIR filter (b) equivalent MDFG

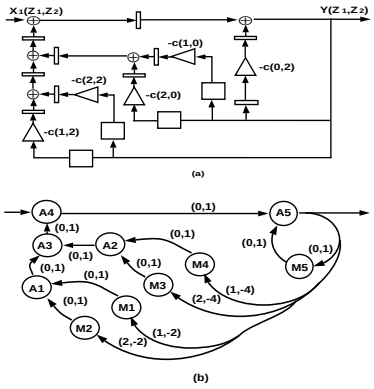


Figure 5: (a) circuit design of the retimed filter (b) equivalent MDFG

exists when the retiming function is chosen parallel to one of the outermost dependence edges. If a cycle is detected, a new retiming function will be used. The retiming of each node v_i by LENGTH(v_i) $\times r$ results in final delay vectors of the form $d(e) - j \times r$, $d(e) + j \times r$, or $j \times r$ where $j =$LENGTH(v_i). Theorem 4.4 shows that a realizable graph is achievable. Item (2) is proven by theorem 4.4 and corollary 3.2. Item (3) is immediate from theorem 4.3. □

On the general case, the desired cycle time may be large enough to accommodate the sequential execution of two or more operations. In such a situation, a slight modification of the algorithms, combining those nodes that serially executed would fit in the target cycle time, produces a circuit (not fully parallel) with a lower number of non-zero delay edges, satisfying the specified cycle time. Such a solution is not presented in this paper due to space constraints.

6 Example

In this section we present the application of our method to a 2-D filter design with transfer function

$$H(z_1, z_2) = \frac{1}{\left(1 - \sum_{n_1=0}^{2} \sum_{n_2=0}^{2} c(n_1, n_2) * z_1^{-n_1} * z_2^{-n_2}\right)}$$

with $c(i, j) = 0$ for $j = 1$ and for $i = j = 0$. The design is shown in figure 4. Using the algorithm *Comdr*, we begin by finding a possible MD retiming function. the outermost dependence edges are $(1, 0)$ and $(0, 2)$. The former can not be selected as basis for the MD retiming function due to the cycle $M4 \rightarrow A2 \rightarrow A3 \rightarrow A4 \rightarrow A5$. Therefore, the selected MD retiming function is $r = (0, 1)$. The algorithm produces the following coefficients for the retiming function: $A5$ is assigned to -4, node $A4$ and $M5$ to -3, $A3$ to -2, $A1$ and $A2$ to -1, and finally, $M1$, $M2$, $M3$, and $M4$ to 0. The resulting retiming function is: $r(A5) = (0, -4)$, $r(A4) = r(M5) = (0, -3)$, $r(M3) = (0, -2)$, and $r(A2) = (0, -1)$.

The final fully parallel graph and the modified circuit design can be seen in figure 5. Comparing the results with the original cycle time, we notice that the critical path was reduced from 4 adders and one multiplier to one functional element, with a cycle time equal to the execution time of one multiplication. This represents a gain equivalent to 5 times in computational time. Another important result on this example is that all new delays have size one for the schedule vector $s = (1, 0)$. It is clear that the full parallelism produced by our algorithm will always guarantee optimal results with respect to the cycle time of the circuit.

References

[1] L.-F. Chao and E. H.-M. Sha, " Static Scheduling of Uniform Nested Loops," *Proc. of the 7th Int'l Parallel Processing Symposium* , 1993, pp. 1421-1424.

[2] Gnanasekaran, R., " 2-D Filter Implementation for Real-Time Signal Processing". *IEEE Trans. on Circuits and Systems*, 1988, vol. 35, n. 5, pp. 587-590.

[3] S. Y. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ: Prentice Hall, 1988.

[4] C. E. Leiserson and J. B. Saxe, " Retiming Synchronous Circuitry". *Algorithmica*, 6, 1991, pp. 5-35.

[5] L.-S. Liu, C.-W. Ho and J.-P. Sheu, " On the Parallelism of Nested For-Loops Using Index Shift Method," *Proc. of the International Conference on Parallel Processing*, 1990, Vol. II, pp. 119-123.

[6] N. L. Passos, E. H.-M. Sha, and S. C. Bass, " Schedule-Based Multi-Dimensional Retiming". *Proc. of the 8th International Parallel Processing Symposium*, 1994, pp 195-199.

[7] A. Darte and Y. Robert, " Constructive Methods for Scheduling Uniform Loop Nests," *IEEE Trans. on Parallel and Distributed Systems*, 1994, Vol. 5, no. 8, pp. 814-822.

[8] R. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA, 1983.